Universidad Autónoma de Madrid

Escuela Politécnica Superior

Máster Universitario en Investigación e Innovación en Tecnologías de la Información y las Comunicaciones

TRABAJO DE FIN DE MÁSTER

STUDY ON VIABILITY, STRENGTHS AND WEAKNESSES OF BAG OF LITTLE BOOTSTRAPS (BLB)

Pablo de Viña Carmona
Tutor: Gonzalo Martínez-Muñoz

13 de febrero de 2018

STUDY ON VIABILITY, STRENGTHS AND WEAKNESSES OF BAG OF LITTLE BOOTSTRAPS (BLB)

Autor: Pablo de Viña Carmona
Tutor: Gonzalo Martínez-Muñoz

Machine Learning Group
Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid

13 de febrero de 2018

Agradecimientos

Me gustaría comenzar agradeciendo a todos los compañeros y profesores que he tenido hasta la fecha, que han sido cómplices de que yo entregue este trabajo hoy aquí. También a mi familia, especialmente a mis padres y a mi hermana, por el cariño y el apoyo constante que siempre me han brindado. De la misma manera quisiera agredecer a mis amigos de toda la vida y en general a todas aquellas personas que me han permitido crecer y descubrirme.

Pero sobretodo me gustaría agradecer a Gonzalo Martínez-Muñoz haber sido un magnífico tutor ayudándome en todo aquello que estaba en su mano; su ayuda y apoyo han sido incalculables durante el desarrollo de este proyecto.

Abstract

Abstract — Randomization techniques represent a fundamental tool used to improve the performance of many machine learning techniques. These techniques aim to generate many simple solutions of the problem in question to later combine them obtaining a more precise solution. Some of them are Bagging, Random Forest and most recently developed Bag of Little Bootstraps (BLB).

We know that the total amount of data in the world was 4.4 zettabytes in 2013 and that is set to rise steeply to 44 zettabytes by 2020. It is clear that the world is changing and the methods we use to analyze it must do as well; what might be efficient for a dataset of a given size becomes unmanageable for another dataset of larger size.

For this reason, researchers at the University of California developed a method to work with much larger data sets than usual: the Bag of Little Boostraps. This method does not intend to analyze the data in a significantly different way to the traditional bagging, but is intended to be a modification of this much more efficient than the original.

In this paper we will study the BLB and its main characteristics: its effectiveness as an automatic learning method, its strengths, its weaknesses, as well as the conditions for a dataset to be a good candidate to be analyzed with this method. Finally, we will discuss possible improvements of the method as well as possible ways of working to follow.

Key words — machine learning, bag of little boostraps, BLB.

Resumen

Resumen — Las técnicas de aleatorización representan una herramienta fundamental utilizada para mejorar el rendimiento de muchas técnicas de aprendizaje automático. Estas técnicas pretenden generar múltiples soluciones simples del problema en cuestión para luego combinarlas obteniendo una solución más precisa. Algunas de las técnicas de este tipo son Bagging, Random Forest y más recientemente desarrollado Bag of Little Bootstraps (BLB).

Sabemos que la cantidad total de datos en el mundo fue de 4.4 zettabytes en 2013 y que está previsto que aumente abruptamente a 44 zettabytes en 2020. Está claro que el mundo está cambiando y los métodos que utilizamos para analizarlo deben hacerlo también; lo que podría ser eficiente para un conjunto de datos de un tamaño dado se vuelve inmanejable para otro conjunto de datos de mayor tamaño.

Por esta razón, los investigadores de la Universidad de California desarrollaron un método para trabajar con conjuntos de datos mucho más grandes de lo habitual: Bag of Little Bootstraps. Este método no pretende analizar los datos de una manera significativamente diferente al bagging tradicional, sino que pretende ser una modificación de este mucho más eficiente.

En este trabajo se estudiará el BLB y sus principales características: su eficacia como método de aprendizaje automático, sus puntos fuertes, sus debilidades, así como las condiciones para que un conjunto de datos sea un buen candidato para ser analizado con este método. Finalmente, se plantearán posibles mejoras del método así como posibles vías de trabajo a seguir.

Palabras clave — aprendizaje automático, bag of little boostraps, BLB.

Índice general

Índice de figuras

1

Introducción

Hace unos años los ordenadores eran unas máquinas del tamaño de una habitación al alcance de unas pocas empresas, sin embargo hoy en día es extraño encontrar un hogar que no tenga al menos media docena de dispositivos de computación: PC's, portátiles, smartphones, smartwatches, etc.

Además de la cantidad de dispositivos y de la accesibilidad de los mismos, también ha aumentado radicalmente su potencia; por ejemplo, en el año 1996 el Hitachi SR2201/1024 situado en la Universidad de Tokio en Japón batió un record al ser capaz de alcanzar los 220.4 GFLOPS en un test de rendimiento (los FLOPS son una medida de rendimiento de computación y significa "operaciones de coma flotante por segundo"); hoy en día sólo la GPU de un PC doméstico puede superar los 10000 GFLOPS.

Todo esto sumado a la gran cantidad de información de la que disponemos (y la que se genera cada día) ha provocado la necesidad de desarrollar técnicas que permitan a las computadoras aprender; de forma que además de ser rápidas, sean inteligentes. Al campo de las ciencias encargado de ello se le demonina aprendizaje automático (del inglés, "Machine Learning").

El término «machine learning» fue acuñado en 1959 por Arthur Samuel, un pionero de la inteligencia artificial, mientras trabajaba en IBM [1]. El principal objetivo del aprendizaje automático es permitir a los ordenadores aprender las reglas que subyacen en un conjunto de datos, con el objetivo de poder hacer predicciones posteriormente. A lo largo de los años han ido surgiendo distintos métodos con éste objetivo: árboles de decisión, redes neuronales artificiales, máquinas de vectores de soporte, clústering, etc; cada uno de ellos tiene sus ventajas y sus inconvenientes frente al resto pero todos ellos persiguen lo mismo, generalizar comportamientos y aprender reglas a partir de información suministrada en forma de ejemplos.

Con estos algoritmos se han desarrollado modelos capaces de aprender multitud de problemas; desde jugar a las tres en raya o a las damas en sus inicios, hasta vencer a los campeones del mundo de otros más complejos como el ajedrez o el Go.

Para poder hacer esto, se parte de una base de ejemplos con una serie de parámetros de entrada y que pueden tener además las salidas deseadas del sistema (aprendizaje supervisado) o no tenerlas (aprendizaje no supervisado). Sobre estos ejemplos se aplican los métodos para tratar de aprender las reglas que subyacen en los datos, con el fin de poder realizar predicciones partiendo de ejemplos nuevos.

1.1. Alcance

En este trabajo vamos a estudiar brevemente algunos de los algoritmos de aprendizaje más utilizados, para posteriormente dar paso al estudio en mayor profundidad de uno de ellos, el Bag of Little Bootstraps o BLB. Algunos de los objetivos a cubrir en este trabajo son:

- Estudio del estado del arte. Breve análisis de algunas de las técnicas de aprendizaje automático más comunes.
- Descripción y análisis del BLB. Explicación en mayor profundidad del algoritmo que se pretende estudiar.
- Estudio del comportamiento del algoritmo en función de los valores de sus hiperparámetros. Se realizarán cálculos con distintos conjuntos de datos y parametrizaciones y se analizarán sus resultados.
- Analizar posibles limitaciones y mejoras del BLB. Una vez estudiado y comprendido el BLB, se presentarán algunos puntos débiles del mismo así como ideas para mejorarlo.

1.2. Estructura del documento

Este documento se ha intentado dividir de forma que un lector con pocos conocimientos en la materia pueda profundizar en el campo del aprendizaje automático de forma sencilla.

Tras una breve introducción en el capítulo 1, en el capítulo 2 se presentan algunos de los algoritmos de aprendizaje automático más utilizados.

En el capítulo 3 se detalla cómo se ha codificado el algoritmo, cómo se han realizado las ejecuciones, qué aproximaciones se han tomado, etc; mientras que en el capítulo 4 se describen brevemente los conjuntos de datos estudiados (en el apéndice se describen con mayor profundidad).

Posteriormente, en el capítulo 5, se presentan los resultados de las ejecuciones para los distintos conjuntos de datos. Se muestran gráficas comparativas, datos numéricos, y se presentan algunas conclusiones obtenidas.

Finalmente, en los capítulos 6 y 7 se resumen las conclusiones obtenidas así como las posibles líneas de investigación a seguir.

2

Estado del Arte

Como hemos comentado en la introducción, el objetivo del aprendizaje automático es aprender las reglas que subyacen en un conjunto de datos con el objetivo de poder hacer predicciones posteriormente. Para ello, partiendo de unos ejemplos base se aplican algoritmos de aprendizaje automático como árboles de decisión, redes neuronales artificiales, máquinas de vectores de soporte o redez bayesianas.

Sin embargo, a la hora de modelizar la información surge un problema y es que el aprendizaje puede verse demasiado condicionado por los datos concretos estudiados. Al tratar de realizar predicciones a partir de las reglas aprendidas pero partiendo de otros datos los resultados pueden no ser muy buenos debido a que el modelo se ha ajustado demasiado a las particularidades de los ejemplos que le han sido mostrados anteriormente. A este efecto se le denomina sobreajuste ("overfitting").

Una de las técnicas que se pueden aplicar para poder realizar mejores predicciones en conjunción con los algoritmos nombrados anteriormente es el boostrap [2].

2.1. Bagging

La agregación bootstrap (o bagging) es un meta-algoritmo diseñado para mejorar la estabilidad y precisión de técnicas de aprendizaje automático [2] [3]. Este meta-algoritmo busca mejorar la precisión de predictores (algoritmos de aprendizaje automático) escogiendo distintas muestras para el entrenamiento y creando un predictor para cada una. El predictor final será la media de estos si el objetivo es predecir un valor numérico o la clase más votada si el objetivo es categórico.

El procedimiento para trabajar con muestras bootstrap es el siguiente: se parte de un conjunto de datos de entrenamiento $D = \{(\mathbf{x}_i, y_i) \text{ con } i = 1 \dots N\}$, donde y_i es la variable objetivo que tratamos de predecir, y $\mathbf{x}_i$ son las variables de los ejemplos que usaremos para conseguir éste propósito. A partir de este conjunto D se generan m nuevas muestras D_i de tamaño N'. Cada una de las muestras se crea extrayendo aleatoriamente ejemplos de D. Estos ejemplos pueden volver a introducirse en el conjunto original de forma que puedan ser extraídos múltiples veces (muestra bootstrap con reemplazamiento); o pueden apartarse de forma que cada ejemplo pueda aparecer una vez o ninguna en el conjunto final (muestra bootstrap sin reemplazamiento). Por lo tanto, al generar una muestra bootstrap con reemplazamiento D_i será probable que encontremos algunos ejemplos de D repetidos y que otros no se encuentren en ella. En el caso particular (y común) de que $N' = N$, para un valor suficientemente alto de N el número de ejemplos únicos tendrá un valor de media $(1 - \frac{1}{e}) \approx 63.2\,\%$. Esto quiere decir que para un valor grande de N el $36.8\,\%$ de los ejemplos estarán repetidos y el resto serán únicos.

Una vez creadas las muestras bootstrap con reemplazamiento, se construyen m árboles (uno por cada muestra) y se combinan promediando para regresión o mediante voto por mayoría para clasificación. De esta manera lo que se consigue es simular una cantidad de datos para el entrenamiento mucho mayor, creando conjuntos de datos a partir de alteraciones del que ya tenemos. Así entrenamos con variaciones que podrían darse realmente consiguiendo más estabilidad y precisión y evitando caer en el sobreajuste [4].

Se ha comprobado que este sistema funciona mejor para procedimientos inestables, es decir, cuando unos pequeños cambios en la muestra producen grandes cambios en el predictor. Cabe destacar que no estamos creando ejemplos nuevos en las muestras de entrenamiento, sino que estamos aumentando el peso de unos y reduciendo el de otros; debido a esto, si el proceso fuera estable apenas generaría diferencias entre los árboles, mientras que si el proceso es inestable esos pequeños cambios provocarían una gran diversidad de árboles que es lo que necesitamos para que el algoritmo funcione bien.

En la publicación estudiada [3] se hicieron pruebas en varios conjuntos de datos muy diferentes (la mayoría provenientes del repositorio UCI) y las muestras se crearon usando 50 divisiones aleatorias del set completo; este proceso se llevó a cabo 100 veces para cada conjunto de datos. Los resultados son muy concluyentes: en torno a un 20-47 % de reducción del error en la clasificación con respecto a un clasificador único.

Esta mejora en el error se debe a que la agregación de muchos clasificadores puede reducir las limitaciones de los clasificadores base que se combinan.

2.1.1. Tamaño óptimo del bagging (estimaciones oob)

Como hemos mencionado anteriormente, el valor generalmente utilizado en bagging es $N = N'$ pero este valor no tiene por qué obtener los mejores resultados; es posible mejorar los resultados del bagging escogiendo otro tamaño de muestra diferente al estándar [5]. De manera general, el menor valor del error se encuentra cogiendo muestras bootstrap

más pequeñas que la original: menor que N para bagging con reemplazamiento y menor que $\frac{N}{2}$ para bagging sin reemplazamiento. Sin embargo, los valores óptimos del tamaño de las muestras pueden ser muy diferentes según el problema al que nos enfrentemos, por lo que no debemos tratar de calcular un valor óptimo único sino una manera de estimarlo dado un problema de clasificación concreto.

Para ello pueden emplearse estimaciones out-of-bag (oob) del error; es decir, calcular la etiqueta de una instancia empleando las predicciones de clasificadores que no emplearon esa instancia para entrenar. El error out-of-bag se calcula como la fracción de los ejemplos del conjunto de entrenamiento original que han sido etiquetados incorrectamente empleando esas predicciones. En la publicación [5] se estudian muchos tipos de problemas y se concluye que la elección de $N'_{wr} = N$ y $N'_{wor} = \frac{N}{2}$ no suele ser la más adecuada en cuanto al error de clasificación, siendo N'_{wr} y N'_{wor} el tamaño de las muestras con y sin reemplazamiento respectivamente. El error en conjuntos construidos entrenando clasificadores individuales con muestras bootstrap muy pequeñas es muy pobre en general. De hecho, a medida que el ratio de muestreo tiende a cero, la precisión del conjunto se acerca a la fracción de ejemplos que pertenecen a la clase mayoritaria.

Para un gran rango de tamaños de muestreo intermedios los conjuntos mejoran el resultado del bagging estándar. Si cogemos las muestras demasiado grandes, el error presenta una tendencia a aumentar. De hecho, en el muestreo con remplazamiento y para un valor suficientemente grande tendríamos todos los ejemplos en todas las muestras por lo que los árboles entrenados serían muy similares entre sí. En general, se comprueba que cogiendo un ratio de un 80 % (para el caso de muestreo con reemplazamiento) el error se reduce considerablemente en bastantes problemas y se obtiene un valor similar en otros; pero en ningún caso empeora significativamente. Para un ratio de 120 % se obtiene un error similar en muchos casos y peor en otros, pero nunca mejor.

Por último, si bajamos del umbral del 69.3 % (estadísticamente menos de la mitad de los ejemplos originales se cogen en cada muestra bootstrap), aumentamos el número de casos en los que mejora el error, pero también el número de casos en los que empeora. Para el muestreo sin reemplazamiento los resultados son muy parecidos, para un ratio $\frac{4}{9}$ (ligeramente por debajo del $\frac{1}{2}$ estándar) los resultados son iguales a o mejores en casi la totalidad de los problemas. Si escogemos un ratio superior a $\frac{1}{2}$ los resultados se mantienen parecidos o empeoran, pero en ningún caso mejoran. Por último, si reducimos la muestra por debajo de $\frac{4}{9}$, nos encontramos con más casos en los que mejora, pero también más en los que empeora el error.

Cabe destacar que en los problemas estudiados emplear la estimación oob del tamaño de las muestras nunca supuso un empeoramiento significativo del error generalizado respecto a la elección estándar. Como conclusión final, realizar bagging tomando el tamaño de re/submuestreo estándar mejora los resultados, pero éstos no son nada óptimos. Con tan sólo emplear la estimación oob (out-of-bag) del error generalizado, mejora significativamente en muchos casos el error y además se reduce el coste computacional al tratarse muestras menores que la inicial.

2.1.2. Bagging en vecinos próximos

Vecinos próximos es un método de clasificación supervisada que se basa en clasificar los ejemplos empleando otros cercanos al objetivo en el espacio de los elementos. Para aplicar bagging sobre éste método hay que tener en cuenta algunos detalles, ya que como comentamos anteriormente, el algoritmo de bagging no está pensado para ser aplicado a técnicas de aprendizaje automático que sean estables, como es el caso de vecinos próximos; de hecho realizar bagging usando Knn con k=1 es equivalente a vecinos próximos [3]. Bajo ciertas condiciones, se ha comprobado que bagging reduce la tasa de error de vecinos próximos [6].

Para que funcione eficientemente es muy importante que el tamaño del remuestreo cumpla ciertas condiciones. En [6] se demuestra que si $\lim_{N\to\infty} \frac{N'}{N} = 0$ (siendo N' el nuevo tamaño y N el tamaño anterior) el error converge al error de Bayes. En general, es necesario que el cociente $\frac{N'}{N}$ sea inferior a 0.69 o 0.5 para los casos con o sin reemplazamiento respectivamente.

La necesidad de imponer estas condiciones proviene del hecho de que vecinos próximos es un proceso estable; escogiendo un valor de remuestreo de $N' < 0.69N$ para el caso con reemplazamiento y de $N' < 0.5N$ para el caso sin reemplazamiento convertiremos el proceso en inestable; ya que de esta forma cada ejemplo va a estar en promedio en menos de la mitad de los Knn del conjunto, eliminando la mayoría inicial y provocando cambios en el resultado. Escoger valores de remuestro mayores a los indicados provocaría que el proceso siguiera siendo estable y sería equivalente a a crear un único modelo de vecinos próximos.

En los ejemplos estudiados en la publicación el valor óptimo de N' es siempre mucho menor que N. En general, el valor óptimo se puede estimar mediante validación cruzada.

2.2. Árboles de decisión

El aprendizaje basado en árboles de decisión es un método muy empleado por su reducido coste computacional y su fácil interpretación. Como todos los métodos tiene sus limitaciones y puede no ser la más adecuada para todo tipo de problemas; sin embargo si lo combinamos con otra serie de técnicas se vuelve muy potente, permitiéndonos lograr muy buenos resultados en un tiempo y con un coste computacional reducidos.

El método consiste en crear divisones de los datos en función del valor de una o más variables, de forma que separen los datos de la mejor forma posible. Estas divisiones se producen en los nodos, y tras cada división pueden generarse más nodos con nuevas divisiones hasta que finalmente obtenemos un diagrama con forma de árbol, donde las hojas representan los posibles valores de la variable objetivo.

Podemos ver un ejemplo de árbol de decisión para un problema de clasificación en la (figura 2.1).

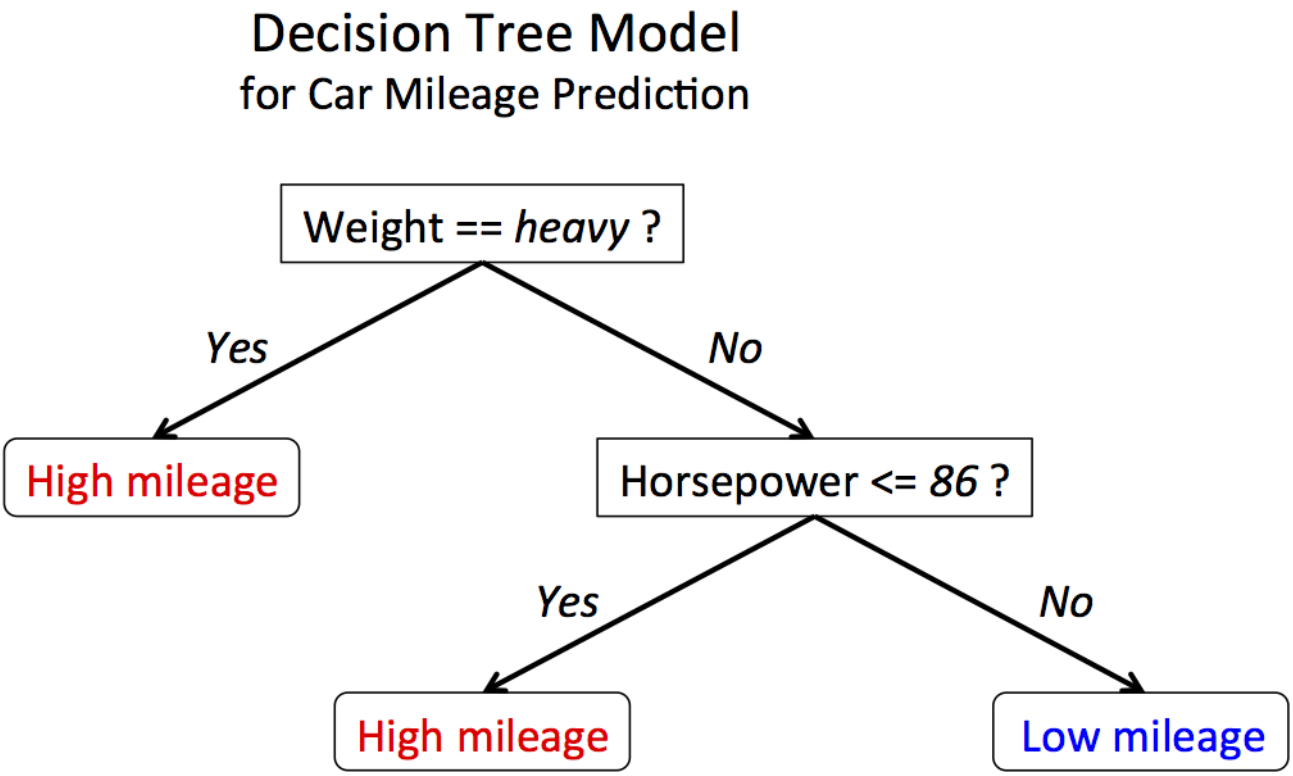


Figura 2.1: Ejemplo de un árbol de desición. Fuente: https://databricks.com/blog/2014/09/29/scalable-decision-trees-in-mllib.html

Existen muchos algoritmos específicos para generar un árbol de decisiones (CART, ID3, C4.5, etc), la diferencia entre ellos es el criterio que siguen para generar los nodos y sus divisiones.

Por ejemplo, el algoritmo CART utiliza la impureza de Gini como métrica para determinar las mejores divisiones. La impureza de Gini mide cómo de habitual sería que un elemento del conjunto elegido aleatoriamente fuera etiquetado incorrectamente si se etiqueta de manera aleatoria de acuerdo a la distribución de etiquetas del subconjunto.

2.3. Boosting

Al igual que bagging, existen otros meta-algoritmos basados en la combinación de clasificadores débiles para obtener uno más robusto. Uno de ellos es el boosting [7], cuya principal diferencia con respecto a bagging es que los subconjuntos de datos para el entrenamiento no se forman de forma aleatoria e independiente. A medida que se van incluyendo clasificadores débiles, aumentan los pesos de los ejemplos que han sido mal clasificados por los modelos anteriores y van formando nuevos clasificadores débiles que finalmente se combinarán para dar lugar a uno más robusto. Denominaremos clasificadores débiles a aquellos que estén sólo ligeramente correlacionados con la clasificación real, y fuertes a aquellos que estén fuertemente correlacionados.

Un ejemplo de algoritmo de boosting es el AdaBoost (Adaptative Boosting). Este parte de una serie de expertos que realizan unas predicciones o siguen ciertas estrategias para ganar. El objetivo final es minimizar la pérdida total, para ello encuentra una serie de límites para el valor de ésta pérdida, con el objetivo de tener un valor "no mucho peor" que el del mejor experto. Para lograr esto el algoritmo da un peso inicial a las predicciones y va alterando esos valores según acierten o no estos predictores. A diferencia de otros algoritmos de boosting, en vez de votar por mayoría éste también combina las hipótesis más débiles y sólo entonces realiza una predicción. Es decir, en vez de dejar que elijan los mejores por votación, permite a todos los clasificadores que voten y obtiene el resultado final mediante ponderación. Uno de los puntos fuertes de este algoritmo es que funciona incluso si se desconoce completamente todo lo relativo al problema en cuestión (incluso por ejemplo si se desconoce la matriz que representa la pérdida en función de la entrada y la salida).

En general, este algoritmo ofrece muy buenos resultados frente a otras técnicas como el bagging o la aleatorización, pero es extremadamente sensible al ruido y a los outliers. Incluyendo en los datos un 10 % de ruido, los resultados de este algoritmo son ampliamente peores que los de las otras técnicas mencionadas [8].

Los distintos meta-algoritmos y técnicas que hemos visto pueden aplicarse sobre multitud de modelos de aprendizaje automático. Nosotros vamos a centrarnos en su aplicación sobre árboles de decisión y los distintos métodos que los utilizan.

2.4. Random Forest

Los árboles de decisión pueden combinarse para crear otro algoritmo llamado Random forest. Este es una combinación de árboles predictores tal que cada árbol depende de los valores de un vector aleatorio creado independientemente y con la misma distribución para cada uno de estos. El método combina la idea de bagging y la selección aleatoria de atributos para construir una colección de árboles de decisión con variación controlada [9]. Definiremos F como el número de variables aleatorias sobre las que se buscará la mejor división en cada rama nueva de los árboles. Existen varios métodos para generar random forests: Forest-RI, donde el valor de F puede tomar dos valores, $F = int(\log_2 M + 1)$ donde M es el número de variables que tenemos inicialmente; y Forest-RC, que se basa en hacer separaciones utilizando combinaciones lineales de variables. Existen muchos posibles valores de F. En el ejemplo de la publicación estudiada, en el caso de Forest-RI el valor óptimo de F se sitúa cerca de 4; y en Forest-RC se obtiene el menor error para F=1, por la relación entre fuerza y correlación; en cambio para conjuntos de datos muy grandes continúa decreciendo para valores de F más grandes, como F=25. En general, los resultados no dependen fuertemente de F, excepto para grandes conjuntos de datos [9].

Sin embargo, si analizamos conjuntos de datos pobres (poco diferenciados entre las distintas clases) comprobamos Forest-RI con F=1 no converge pasadas 2000 iteraciones; mientras que con F=10 sí que lo hace, dando un error del 3 %. Con F=25 el error se reduce a 2.8 %. En general, en muchos problemas el rendimiento del algoritmo Random forest es

muy similar al del boosting y es más simple de entrenar y ajustar. Como consecuencia Random forest es popular y es ampliamente utilizado. Otra de las ventajas de este método frente a otros es que resulta muy sencillo realizar estimaciones del error generalizado, fuerza, la correlación, etc., empleando la técnica out-of-bag explicada anteriormente.

Por último, Random forest también nos permite obtener fácilmente información sobre las variables. Si escogemos una variable al azar y permutamos sus valores aleatoriamente, observando qué sucede con el error podemos comprobar que para algunas variables el error aumenta considerablemente, y sin embargo con otras apenas cambia. Esto quiere decir que esa variable es muy importante, es decir, que aporta mucha información. Al combinar estas variables, también descubrimos que algunas que parecen aportar información, en realidad sólo repiten la información de otras y no resultan útiles. Este tipo de análisis es muy útil para analizar la naturaleza de los problemas ya que nos da una idea de las variables que realmente tienen peso sobre la clasificación. Como conclusiones, comparado con lo visto anteriormente, Random forest es tan bueno como Adaboost y a veces mejor; pero es mucho más robusto frente a outliers y frente al ruido, también es más rápido que bagging [8]. Además es sencillo de paralelizar y las estimaciones out-of-bag son fáciles de calcular y de gran utilidad.

2.5. Bag of Little Bootstraps (BLB)

Como hemos visto anteriormente, el bagging aplicado sobre árboles de decisión proporciona un método simple y potente para obtener estimadores de calidad, sin embargo en datasets grandes presenta un coste computacional alto y lineal con el número de clasificadores. Debido a la naturaleza del bagging, si la muestra es muy grande pero no hacemos más grande el árbol, acabaría dando peores resultados ya que tendería a hacer los mismos árboles una y otra vez. Por ello, la eficiencia de éste método decae si aumenta mucho el tamaño del training set; a diferencia de la aleatorización por ejemplo, que no depende del tamaño [8]. Una alternativa cuando tratamos con grandes cantidades de datos es el Bag of Little Bootstraps (BLB). Este procedimiento tiene un coste computacional mucho más bajo que el bagging estándar, es robusto frente a la elección de los parámetros para conjuntos suficientemente grandes (a diferencia del m out of n bootstrap por ejemplo) y es fácilmente paralelizable (el bootstrap también se puede paralelizar fácilmente, pero surgen complicaciones para manejar los datos cuando las muestras son excesivamente grandes) [10].

BLB funciona combinando los resultados de aplicar bagging sobre pequeñas muestras tomadas del dataset original. El procedimiento a seguir es el siguiente:

Se submuestrea el dataset inicial en s muestras bootstrap de tamaño $b = N^{\gamma} << N$ con o sin reemplazamiento; partiendo de cada una de estas submuestras generamos r muestras bootstrap con reemplazamiento recuperando el tamaño del dataset original (N). Con cada una de estas submuestras de tamaño N entrenamos un árbol de decisión para obtener el estimador de interés. Por último, promediamos estos valores procedentes de cada submuestra para obtener el valor final. Podemos observar un diagrama del BLB en

la figura 2.2.

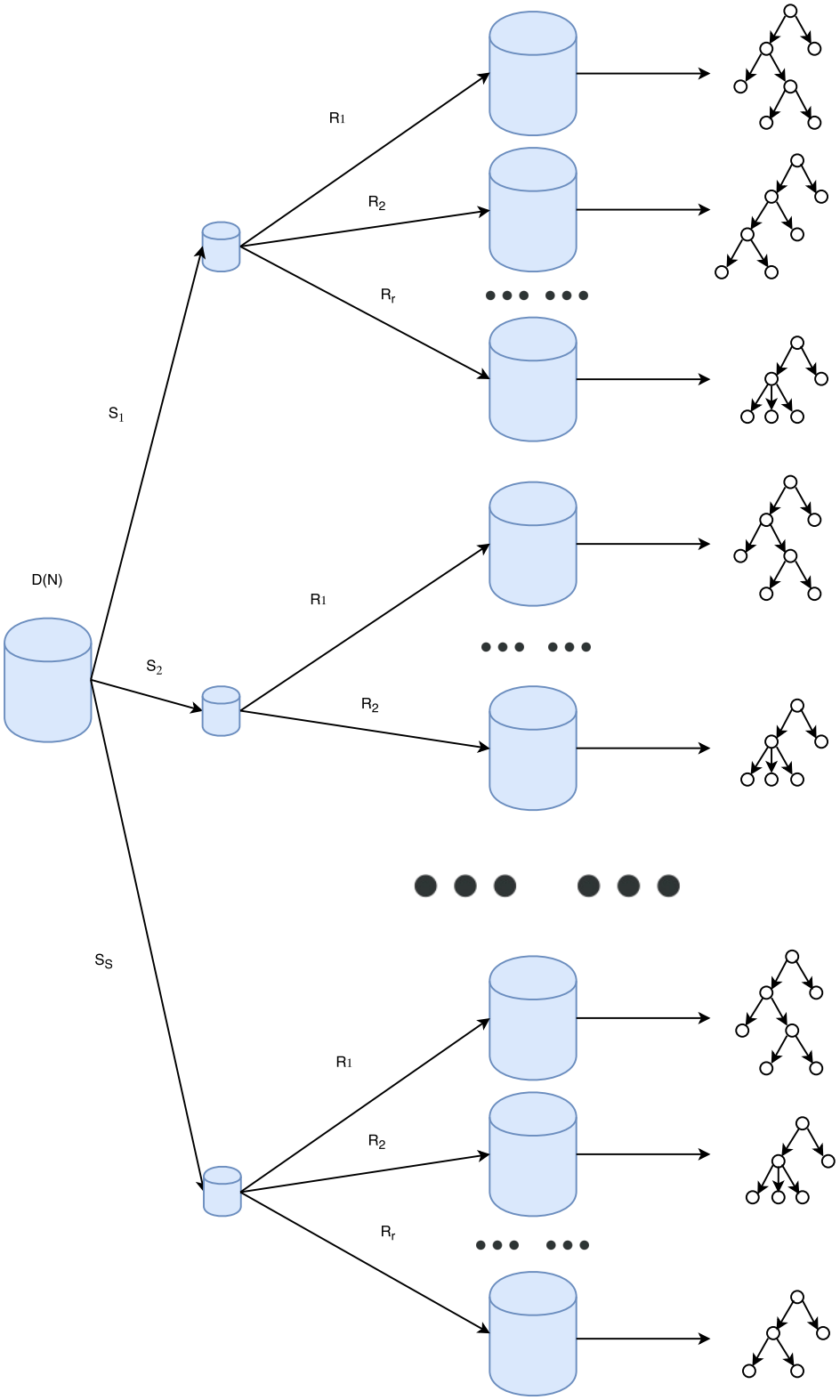


Figura 2.2: Diagrama del funcionamiento del Bag of Little Bootstraps

La clave está en que al final estamos utilizando una y otra vez los mismos datos aunque tengamos muestras de tamaño N, por lo que el coste computacional tanto en cálculo como en lectura y almacenamiento se reduce tremendamente.

Es importante destacar que, aunque es robusto frente a la elección de los hiperparámetros s y r, la elección óptima de estos no es trivial ya que dependerá de la naturaleza de los datos de cada problema. En general estos parámetros se pueden fijar de manera adaptativa para cada caso parando de iterar cuando apenas se produzca cambio entre una iteración y la siguiente; y el tamaño de la muestra b se puede tomar como $b = N^{0.7}$ tal y como se recomienda en [10], aunque como luego veremos este no es generalmente el valor óptimo.

En la bibliografía estudiada [10] se emplean datos sintéticos para estudiar la eficiencia del BLB. Comparado con el bootstrap es capaz de obtener unos resultados similares con menor coste computacional que este. También es mucho más sencillo de distribuir en arquitecturas de computación paralelizada debido al menor tamaño de las submuestras que maneja y es más robusto que el m out n bootstrap frente a la elección del tamaño de las submuestras. Además BLB comparte las propiedades teóricas de bootstrap como la consistencia y la convergencia rápida en las mismas condiciones.

En general, se consigue cumplir las 3 condiciones buscadas: es preciso, automático y escalable.

En nuestro caso, analizaremos los distintos aspectos del BLB empleando árboles aleatorios (o random trees) como clasificadores base del conjunto y que se entrenarán sobre las $r \times s$ muestras creadas.

3

Implementación y descripción de los experimentos

En el apartado anterior se ha descrito el algoritmo de BLB; ahora procederemos a describir cómo se ha implementado exactamente. Es importante destacar que las ejecuciones se han llevado a cabo realizando 10 validaciones cruzadas de 10 particiones, es decir, dado un conjunto de datos se ha dividido en dos subconjuntos disjuntos de forma que se utilice el 90 % de los datos para entrenamiento y el 10 % para test. Este proceso se repite 10 veces, por lo que al final todos los datos son evaluados una única vez en el conjunto de test y 9 en el de entrenamiento. Si además repetimos el proceso completo 10 veces, el resultado será más estable.

Una vez los datos se han separado en los conjuntos de entrenamiento y test correspondientes, se procede a realizar las ejecuciones del programa con distintos valores de los hiperparámetros: b (tamaño de cada submuestra s, también se puede expresar en función de otro hiperparámetro γ; la relación entre ambos es $b = N^{\gamma}$), s (número de submuestras) y r (iteraciones bootstrap sobre cada submuestra). Los cálculos se han realizado para valores altos de γ, s y r y los resultados de las ejecuciones para el resto de valores intermedios se han calculado a posteriori. Los valores concretos empleados se detallan en la sección de resultados, ya que han variado para los distintos conjuntos de datos.

El primer paso consiste en calcular las clasificaciones para cada uno de los árboles del conjunto de r y s más grandes para cada valor de γ.

Aunque inicialmente se hicieron pruebas empleando bagging y árboles CART para cada una de las iteraciones r, finalmente en los resultados que mostraremos se han empleado random trees como método de aleatorización sobre las submuestras por su rapidez y

buenos resultados.

Una vez se obtienen las predicciones de cada uno de los $|s| \times |r|$ modelos, se itera sobre ellas para ir obteniendo los valores de los errores para cada pareja de valores de s y r escogiendo las clasificaciones correspondientes. De ésta forma sólo es necesario calcular los clasificadores y sus predicciones una vez, para s_{max} y r_{max}, y luego se reutilizan para obtener el resto de parejas r y s.

Para el cálculo de las clasificaciones se codificó un programa en C++ con 4 parámetros de entrada:

- El conjunto de datos a procesar.
- γ: el tamaño de cada submuestra tomada será de N^γ, donde N es el número total de datos de entrenamiento.
- s: el número de submuestras de tamaño N^γ a tomar del conjunto original.
- r: número de iteraciones bootstrap a realizar sobre cada una de las submuestras.

Cada una de las ejecuciones genera una serie de ficheros de salida con los resultados de las mismas. Uno de estos ficheros contiene las clasificaciones de todos los árboles generados ($s \times r$ árboles) por lo que partiendo de este fichero y escogiendo sólo los resultados que correspondan se pueden obtener los errores para todos los valores de s y r. Es decir, partiendo de una ejecución con $s = 10$ y $r = 10$, podríamos obtener el error $s = 4$ y $r = 2$ cogiendo exclusivamente los dos primeros árboles de las 4 primeras submuestras. Repitiendo este proceso para todos los valores de s y r conseguimos simular $s \times r$ ejecuciones realizando sólo una con s_{max} y r_{max}. El programa que filtra las clasficaciones se ha codificado en Matlab por la simplicidad a la hora de trabajar con las matrices resultantes.

Uno de los puntos fuertes de BLB es la capacidad de obtener un valor del error muy competente en un intervalo de tiempo muy pequeño; por ello, además del error al que tienda BLB, tendremos en cuenta el tiempo que tarde en hacerlo frente a los tiempos empleados con otros métodos. Sin embargo, no es posible medir este tiempo de forma directa con el método que hemos empleado ya que, como acabamos de comentar, no realizamos cada ejecución por separado.

Para obtener una estimación del tiempo que llevaría cada ejecución en realizarse hemos medido los tiempos de submuestreo, de remuestreo y de generación del árbol para todos los valores de γ y todos los conjuntos de datos. Asumiendo que el tiempo necesario para generar cada árbol es constante, podemos obtener los tiempos de cada ejecucioón multiplicando los tiempos medidos por la cantidad de submuestreos, remuestreos y árboles generados.

Una vez tenemos los errores y los tiempos de las distintas ejecuciones BLB, los comparamos con las ejecuciones Random Forest para comprobar si existe mejora o no frente a un método más tradicional.

4

Conjuntos de datos estudiados

4.1. Breast Cancer Wisconsin (Original) Data Set

Resumen: Predecir la naturaleza de un tumor desarrollado a partir de células mamarias

Tarea: Clasificación

Ejemplos del conjunto: 699

Atributos: 9 más la clase

Missing values: Sí, 16 de los ejemplos contienen un valor desconocido

Descripción: Los ejemplos de este conjunto describen distintas propiedades observadas en células obtenidas de tumores desarrollados a partir de células mamarias. El objetivo final es predecir, a partir de los atributos observados, si el tumor es benigno o maligno. Estos datos se han ido generando periódicamente, procedentes de los casos clínicos del Dr. Wolberg; entre enero de 1989 y noviembre de 1991; sin embargo, trataremos todos los datos como un único conjunto sin tener en cuenta su fecha de obtención.

4.2. MAGIC Gamma Telescope Data Set

Resumen: Clasificar si una señal detectada por un telescopio Cherenkov (el MAGIC) se corresponde con una radiación Gamma o no

Tarea: Clasificación

Ejemplos del conjunto: 19020

Atributos: 10 más la clase

Missing values: No

Descripción: Los datos a analizar se han generado empleando el método de Montecarlo para simular el registro de partículas gamma de alta energía en un telescopio situado Cherenkov, situado en la tierra, bajo la atmósfera. Este tipo de telescopios observan rayos gamma de alta energía, detectando la radiación que emiten las partículas cargadas producidas en las "duchas" electromagnéticas iniciadas por los rayos gamma. Esta radiación Cherenkov atraviesa la atmósfera y queda registrada en el detector, lo que permite reconstruir las características de la "ducha". La información que se recopila consiste en pulsos generados por los fotones Cherenkov que entran en unos tubos multiplicadores. En función de la energía de la radiación gamma inicial, pueden detectarse desde unos pocos cientos de fotones hasta unos 10000, obteniendo un patrón, denominado imagen de "ducha". Analizando los parámetros de esta imagen se puede discriminar estadísticamente los fotones que han sido producidos por radiaciones gamma, y cuales han sido producidos por "duchas" hadrónicas generadas por rayos cósmicos en las partes más elevadas de la atmósfera de la Tierra. Una vez se ha obtenido la primera imagen, ésta se procesa aplicando PCA (Principal Component Analysis), generando una imagen con forma de elipse y cuyos parámetros vamos a analizar (comúnmente llamados parámetros de Hillas).

4.3. Poker Hand Data Set

Resumen: Predecir manos de póker

Tarea: Clasificación

Ejemplos del conjunto: 1000000

Atributos: 10 más la clase

Missing values: No

Descripción: Los ejemplos de este conjunto de datos describen las 5 cartas de una mano de póker. El objetivo es clasificar las manos en 10 posibles categorías.

4.4. Waveform Database Generator (Version 1) Data Set

Resumen: Predecir el tipo de onda del ejemplo

Tarea: Clasificación

Ejemplos del conjunto: Al ser datos generados sintéticamente, podemos escoger cualquier valor arbitrario. Nosotros realizamos los experimentos con 20000 y con 1000000 datos.

Atributos: 21 más la clase

Missing values: No

Descripción: Se generan datos que se corresponden con 3 posibles ondas, y se les añade ruido (con media 0 y varianza 1) sobre los 21 atributos.

5 Resultados

En este apartado vamos a mostrar los resultados obtenidos para los distintos datasets, en función de los hiperparámetros del BLB escogidos. Sólo se mostraran aquellos datos que resulten relevantes, ya que la cantidad de ejecuciones realizadas ha sido muy elevada.

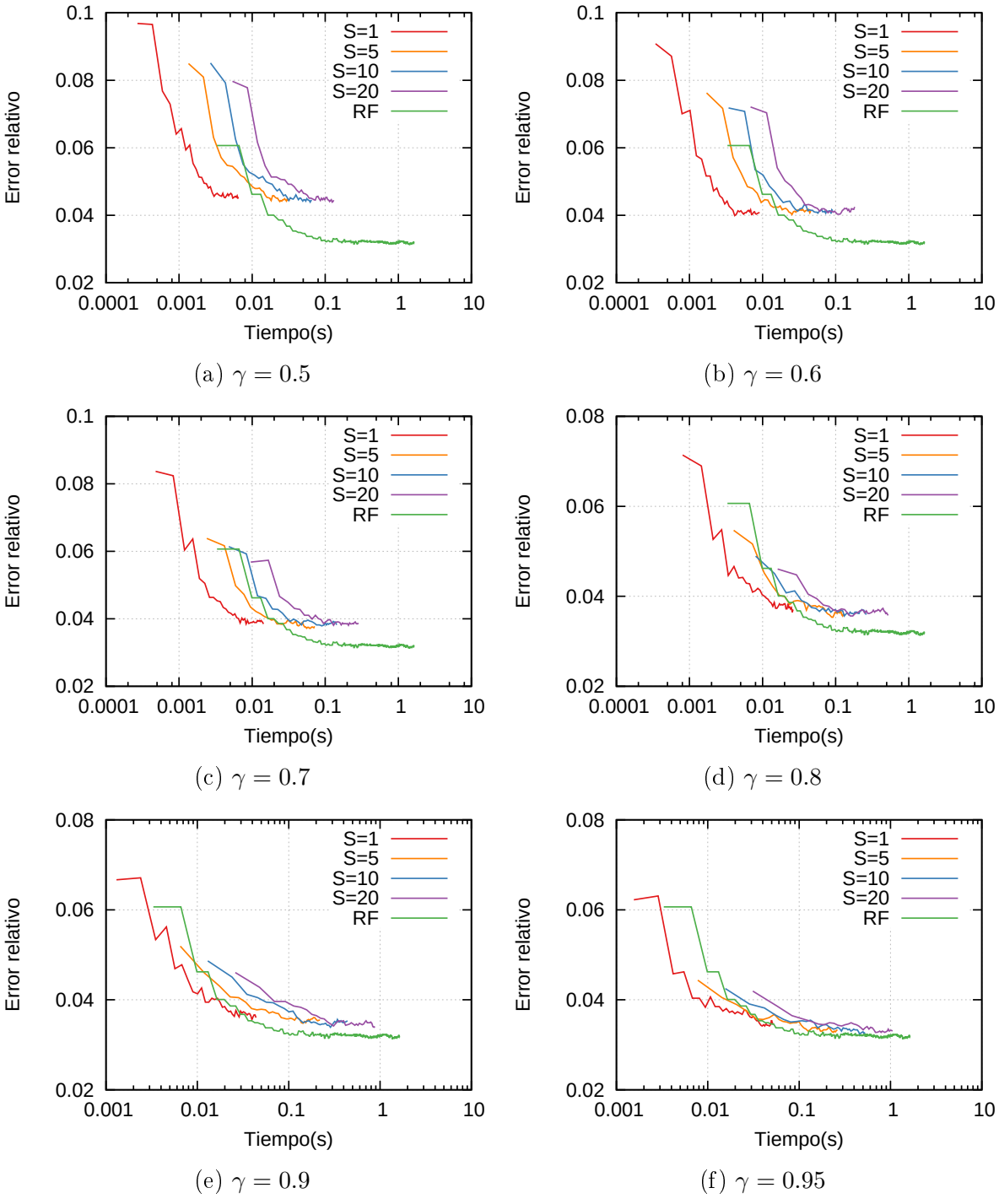
Los tiempos mostrados de las ejecuciones de BLB y Random Forest se corresponden con cada una de las iteraciones de la validación cruzada. Para la mayoría de los casos se han realizado 10 validaciones cruzadas de 10 iteraciones cada una, por lo que el tiempo real empleado en la ejecución sería 100 veces el mostrado. Sin embargo, para los conjuntos de datos más grandes sólo se ha llevado a cabo 1 validación cruzada completa de 10 iteraciones, por lo que para poder comparar los tiempos entre sí, hemos escogido mostrar el tiempo por ejecución.

5.1. Breast

Para este dataset en particular se han realizado los cálculos con los siguientes valores de los hiperparámetros:

$\gamma = \{0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95\}$, $s \in [1, 20]$, $r \in [1, 40]$

Donde recordamos que N^γ es el número de ejemplos de cada submuestra, s es el número de submuestras que se toman de la inicial y r es el número de iteraciones bootstrap que se realizan sobre cada submuestra.



(a) $\gamma = 0.5$ (b) $\gamma = 0.6$

(c) $\gamma = 0.7$ (d) $\gamma = 0.8$

(e) $\gamma = 0.9$ (f) $\gamma = 0.95$

Figura 5.1: Resultados de Breast

En la figura 5.1 podemos observar el error de test obtenido en función del tiempo de entrenamiento; cada línea representa las ejecuciones para distintos valores de r, para un valor fijo de s y γ.

Cada punto representado en estas gráficas representa el error obtenido para una pareja de valores de s y r en una ejecución completa. Los puntos se encuentran unidos entre sí

por líneas para indicar que todas esas ejecuciones independientes comparten el mismo valor de s, pero varían en el valor de r utilizado. De esta forma podremos apreciar para qué valores de los hiperparámetros se obtienen los mejores resultados. En otras palabras, los puntos de cada línea de la figura 5.1 no representan distintos puntos de una misma ejecución sino ejecuciones completamente independientes, pero que comparten el mismo valor de s.

Aunque este sea un conjunto de datos pequeño, observamos que el BLB es capaz de obtener un error inferior al de Random Forest para tiempos pequeños; sin embargo, para valores más altos del tiempo Random Forest parece tender a un valor más bajo del error. Recordemos que el método que estamos analizando se ha presentado como una alternativa a los tradicionales para conjuntos de datos muy grandes, por lo que era de esperar que fuera de ese rango no funcione tan bien como otros métodos. De hecho, para valores bajos de γ como 0.5 o 0.6, las s muestras bootstrap generadas están formadas por tan solo 25 (4.0 %) y 48 (7.6 %) ejemplos respectivamente, por lo que los árboles que se forman de ellos pueden resultar algo pobres.

Comprobamos que a medida que aumenta el valor de γ que empleamos en las ejecuciones del BLB el error obtenido se acerca al obtenido por Random Forest. Finalmente para $\gamma = 0.95$ prácticamente recuperamos el error obtenido por Random Forest, cogiendo en este caso subconjuntos de 456 ejemplos (72.5 %).

Finalmente, para comprobar si es posible alcanzar el error de Random Forest aumentando el tiempo de computación del BLB, hemos realizado ejecuciones seleccionando valores altos de s y r, de forma que estadísticamente acabemos cogiendo todos los ejemplos y se formen muchos árboles de cada submuestra. Los cálculos se han realizado empleando el valor de γ recomendado genéricamente (0.70) y todos los valores de s y r hasta $s_{max} = 50$ y $r_{max} = 30$.

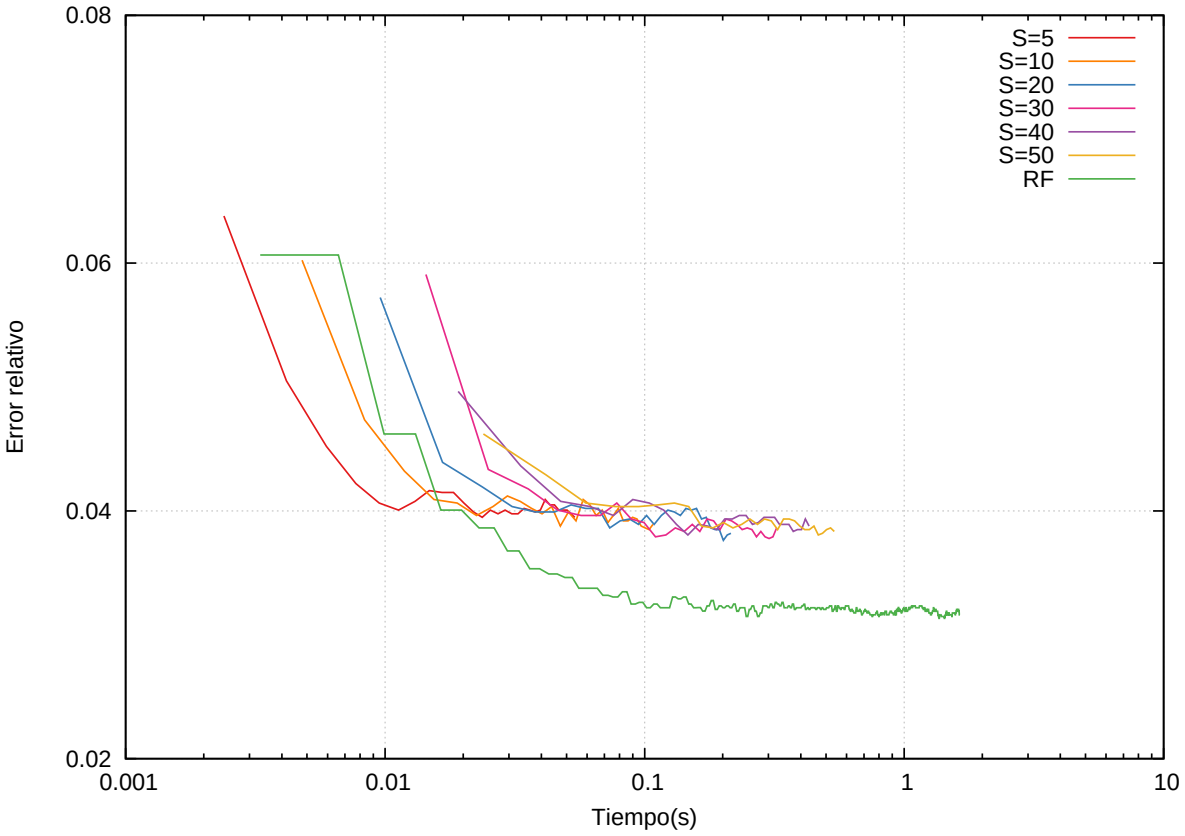


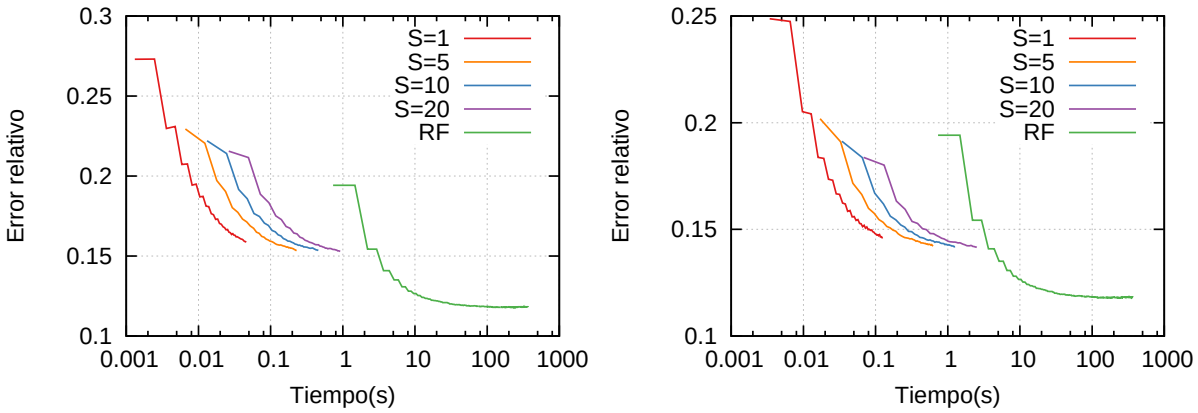
Figura 5.2: $\gamma = 0.70; s_{max} = 50; r_{max} = 30$. Error de test vs tiempo de training (s). Tamaño de submuestra s 173 ejemplos (27.5 %).

El resultado es muy similar al que obtuvimos para $s = 20$; por lo que no parece que baste con aumentar las submuestras y el número de árboles pequeños para igualar el error de Random Forest; los árboles que se generen deben ser «suficientemente buenos» para obtener un buen resultado de la mezcla de todos ellos.

5.2. MAGIC

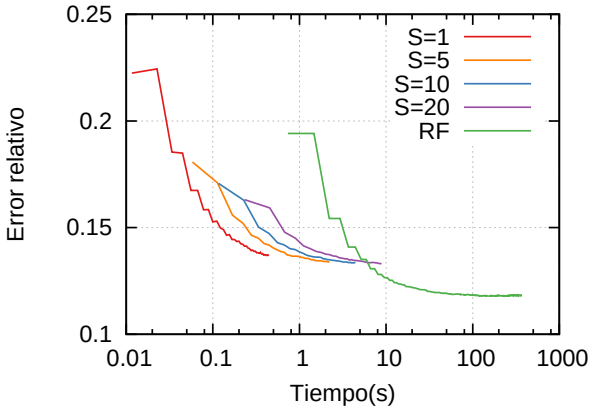
En este apartado presentaremos los resultados del conjunto MAGIC. Estos son los valores de los parámetros que se han utilizado para realizar las ejecuciones:

$\gamma = \{0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95\}$, $s \in [1, 20]$, $r \in [1, 40]$

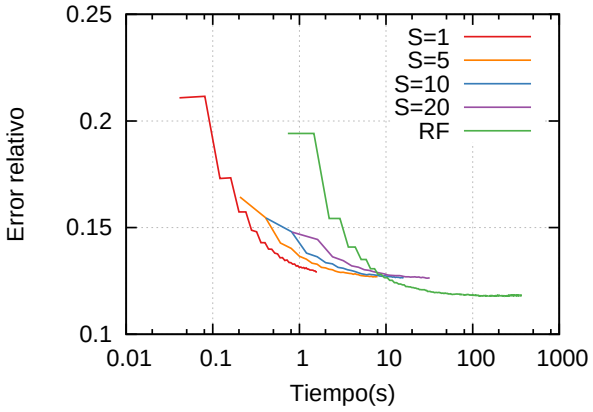


(a) $\gamma = 0.5$

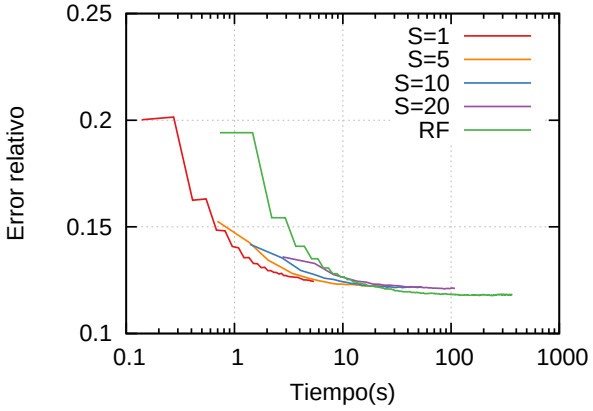
(b) $\gamma = 0.6$



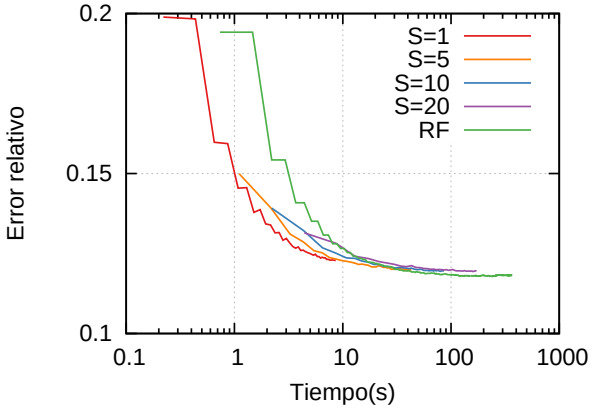
(c) $\gamma = 0.7$



(d) $\gamma = 0.8$



(e) $\gamma = 0.9$



(f) $\gamma = 0.95$

Figura 5.3: Resultados de Magic

En la figura 5.3 podemos observar el error de test obtenido en función del tiempo de entrenamiento para los distintos valores de γ, s y r. El formato de la gráfica es el mismo que el de la figura 5.1. En este caso comprobamos que BLB mejora el error de Random Forest para un tiempo dado de forma mucho más sistemática; obtiene valores razonables del error para tiempos muy reducidos.

Cabe destacar que en todos los casos las curvas de las ejecuciones del BLB comienzan por debajo de la curva de Random Forest, y se mantienen así durante la mayor parte de las gráficas. En aquellas zonas donde esto sucede resulta más eficiente (en términos de tiempo) emplear BLB que Random Forest.

Además, observamos que a medida que aumentamos el valor de gamma mejora el error obtenido; de forma que al alcanzar gamma = 0.95 (61.4 % de ejemplos cogidos en cada muestra) llegamos a obtener un error de 0.1196 recuperándo prácticamente el obtenido con Random Forest (0.1181).

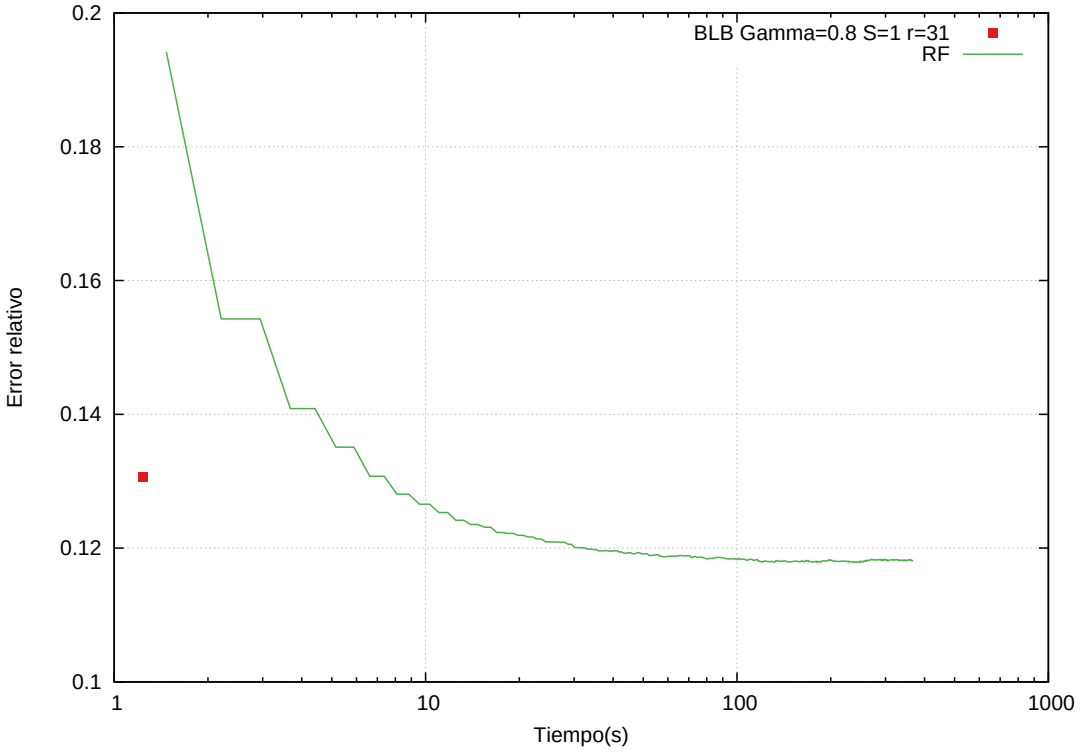


Figura 5.4: Random Forest vs BLB (MAGIC)

En la figura 5.4 podemos observar el error de test obtenido en función del tiempo de entrenamiento de Random Forest frente a una ejecución de BLB con unos hiperparámetros específicos ($\gamma = 0.8$, $S = 1$ y $r = 31$). Se aprecia que para para este volumen de datos ya comienza a ser rentable emplear el BLB, pues obtiene el mismo error que Random Forest tras algunas iteraciones pero en una pequeña fracción del tiempo.

Otro detalle a tener en cuenta es que parece que todas las curvas de las ejecuciones BLB tienden rápidamente a un valor similar, esto quiere decir (tal y como se comenta en

el artículo original) que el método es muy robusto frente a la elección de hiperparámetros (exceptuando γ) incluso para este conjunto de datos que no es especialmente grande.

Para poder evaluar esto teniendo en cuenta las distintos valores de γ y para los distintos conjuntos, vamos a buscar las mejores y las peores ejecuciones de entre todas las ejecutadas y las vamos a representar junto con la ejecución de Random Forest. Definimos mejores ejecuciones como aquellas que consiguen el error más bajo para su tiempo; y peores como aquellas que obtienen el error más alto para su tiempo.

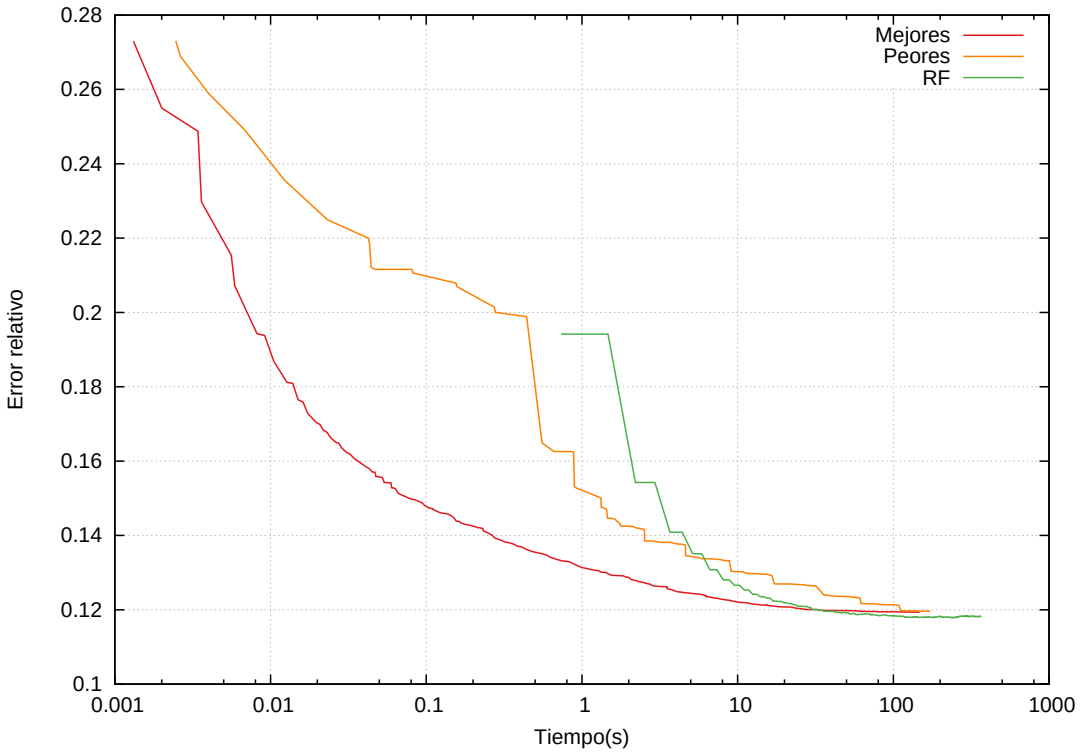


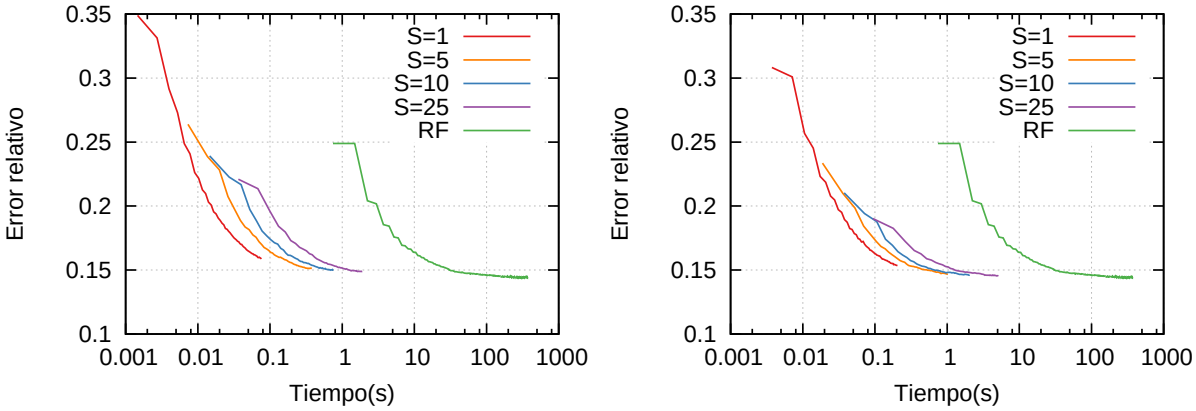
Figura 5.5: Comparativa mejores y peores ejecuciones vs Random Forest (MAGIC)

Comprobamos además que en una buena parte del rango representado incluso las peores ejecuciones del BLB mejoran el error de Random Forest para un mismo tiempo de ejecución. Por otro lado, las mejores ejecuciones del BLB mejoran a Random Forest para casi todo el rango exceptuando para los últimos valores donde ambos obtienen unos errores muy similares aunque ligeramente mejores para Random Forest.

5.3. Waveform 1 (20000 ejemplos)

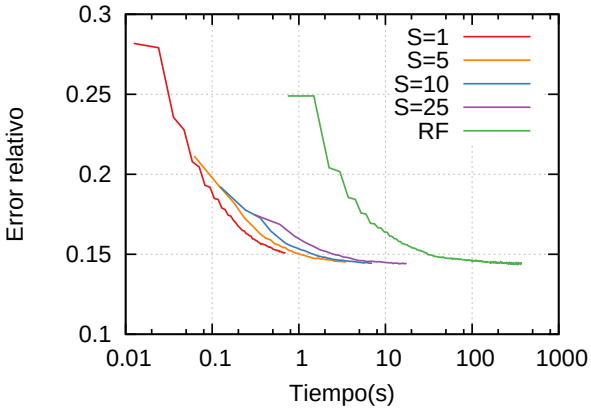
A continuación presentamos los resultados de las ejecuciones de Waveform 1, generado con 20000 ejemplos. Para estas ejecuciones se han utilizado los siguientes parámetros:

$\gamma = \{0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95\}$, $s \in [1, 25]$, $r \in [1, 60]$

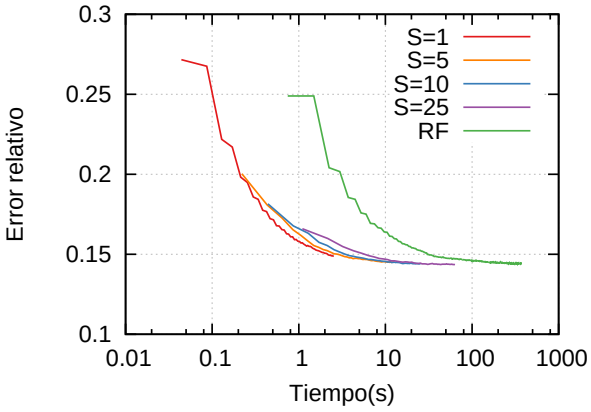


(a) $\gamma = 0.5$

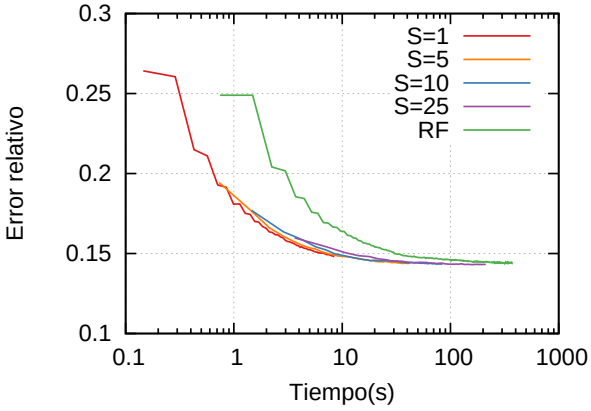
(b) $\gamma = 0.6$



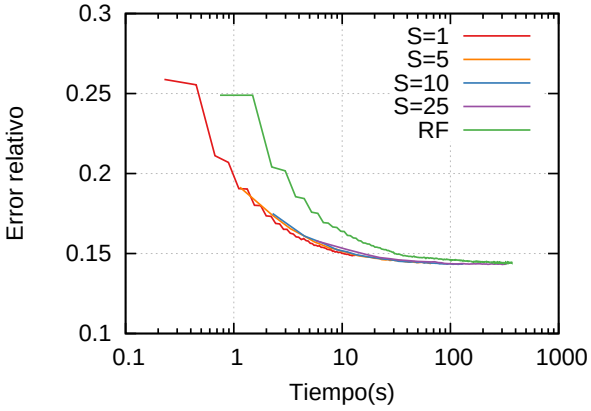
(c) $\gamma = 0.7$



(d) $\gamma = 0.8$



(e) $\gamma = 0.9$



(f) $\gamma = 0.95$

Figura 5.6: Resultados de Waveform (20000 ejemplos)

En la figura 5.6 podemos observar el error de test obtenido en función del tiempo de entrenamiento para los distintos valores de γ, s y r. De nuevo, el formato de la gráfica es el mismo que el de la figura 5.1 y la figura 5.3.

Observamos que para este conjunto de datos, aún siendo de un tamaño muy parecido al anterior, obtenemos un resultado realmente bueno. Para todos los valores de γ el valor obtenido con BLB mejora el obtenido mediante Random Forest.

Por tanto parece que las características propias de cada conjunto de datos sí que afectan al resultado que se obtendrá al aplicar BLB.

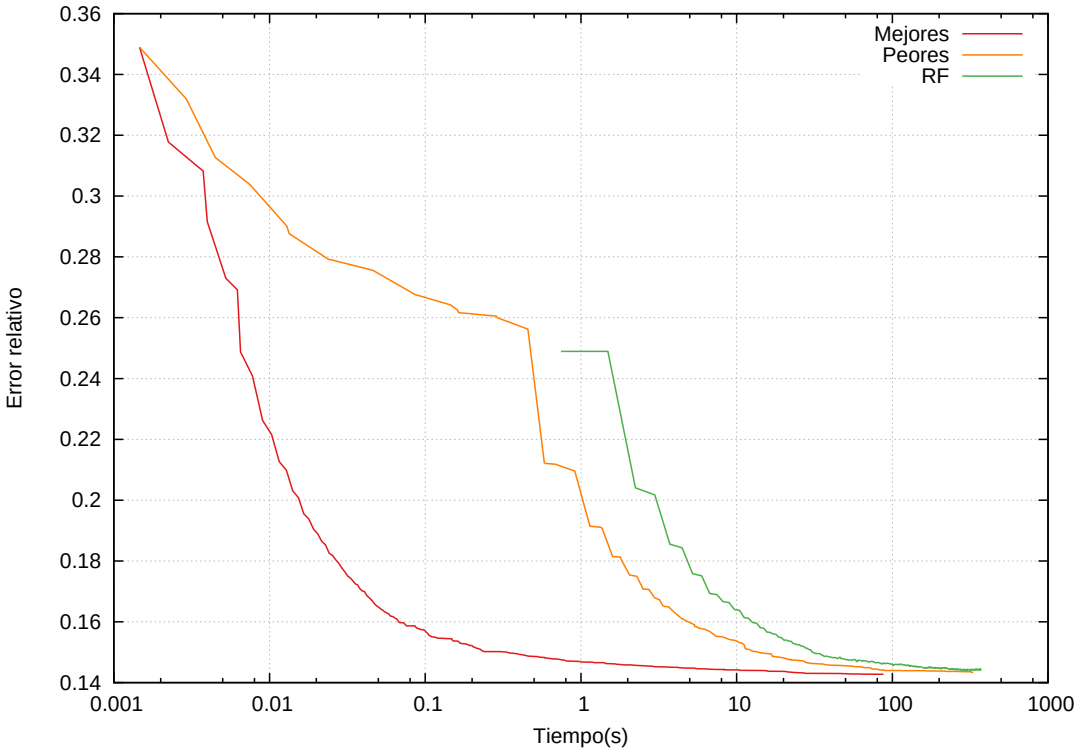


Figura 5.7: Comparativa mejores y peores ejecuciones vs Random Forest (Waveform 20000 ejemplos)

En la figura 5.7 se representan las mejores y las peores ejecuciones, como se describió previamente para la figura 5.5. Analizándola descubrimos que ambas curvas se encuentran más cercanas entre sí que para el conjunto MAGIC y que ambas se encuentran por debajo de la curva de la ejecución de Random Forest. Esto quiere decir que para este conjunto de datos aplicar BLB con cualquier parametrización (dentro de unos márgenes amplios) resulta más eficiente que aplicar Random Forest.

Analizando las mejores ejecuciones para este conjunto, encontramos que casi todas tienen valores muy altos de r, habitualmente entre 50-60. Esto quiere decir que realizar más iteraciones bootstrap r sobre la misma submuestra s reduce más el error para el tiempo que requiere, que emplear un mayor número de muestras s.

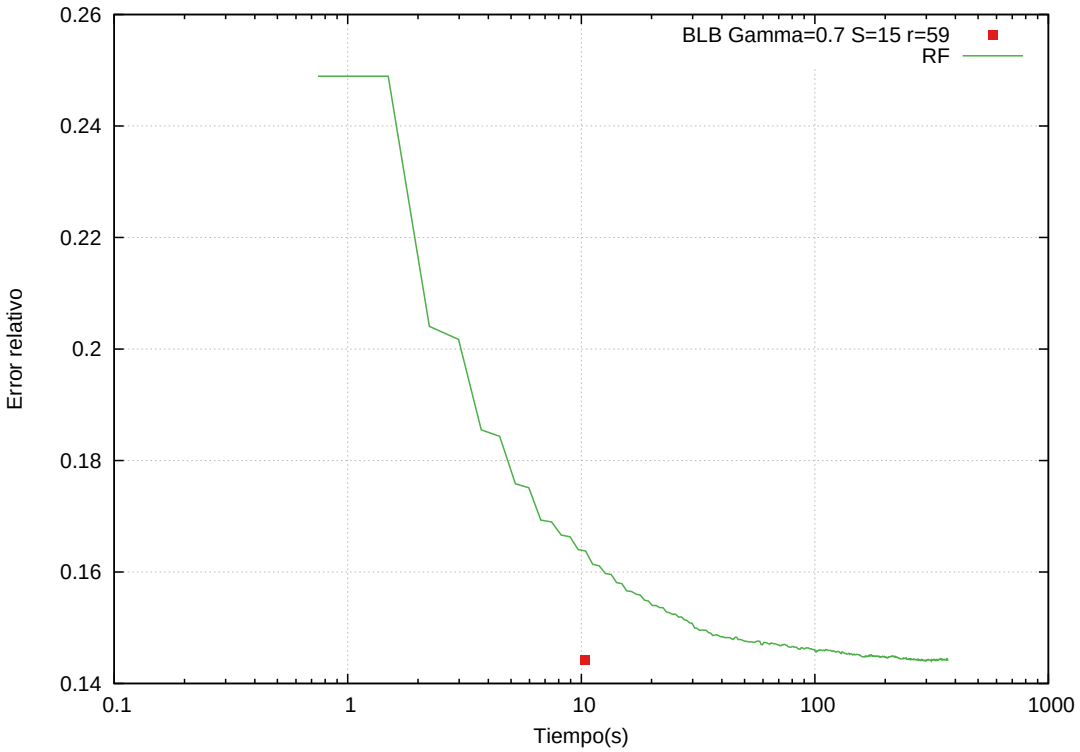


Figura 5.8: Random Forest vs BLB (Waveform 20000 ejemplos)

Por otro lado, escogiendo algunas ejecuciones concretas obtenemos un resultado similar al mostrado en el artículo original [10]. Para la ejecución representada en la figura 5.8, el BLB obtuvo un error de 0.144165 en un tiempo de 10.33 segundos (por iteración CV), mientras que Random Forest acabó obteniendo un error superior (0.144275) tras completar 500 árboles en 372.88 segundos (por iteración CV).

5.4. Poker

En este apartado mostraremos los resultados obtenidos aplicando BLB sobre el conjunto de datos de Poker. Este conjunto es mucho más grande que los anteriores, por lo que hemos reducido los valores de los hiperparámetros con respecto a los utilizados con los conjuntos anteriores ya que se vuelve muy costoso computacionalmente, y para estos conjuntos esperamos que el BLB obtenga mejores resultados en menor tiempo. Estos son los valores escogidos para los hiperparámetros:

$\gamma = \{0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85\}$, $s \in [1, 10]$, $r \in [1, 20]$

En la figura 5.9 podemos observar que los resultados de BLB son mejores que los de Random Forest en todo momento, incluso para la peor elección de hiperparámetros de BLB. Sería interesante estudiar el comportamiento de BLB para valores de los hiperparámetros más elevados, de forma que se pueda comprobar cuál de las dos técnicas tiene una mejor tendencia.

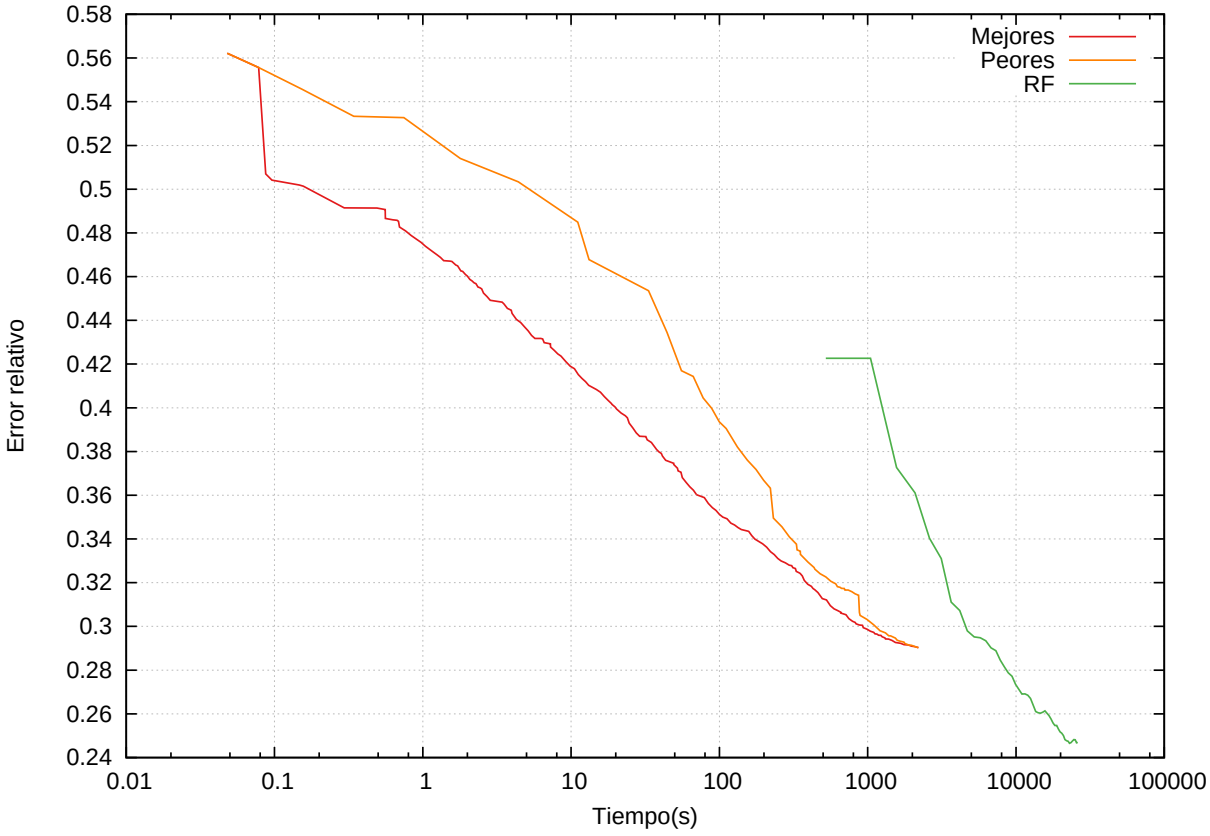


Figura 5.9: Comparativa mejores y peores ejecuciones vs Random Forest (Poker)

5.5. Waveform 2 (1000000 ejemplos)

Este conjunto se ha generado de la misma forma que el Waveform 1 de 20000 ejemplos, pero generando un dataset mucho más grande, de 1000000 de ejemplos. Al igual que para Poker, debido al tamaño del conjunto hemos reducido los valores de los hiperparámetros con respecto a los utilizados con los conjuntos anteriores:

$\gamma = \{0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95\}$, $s \in [1, 10]$, $r \in [1, 20]$

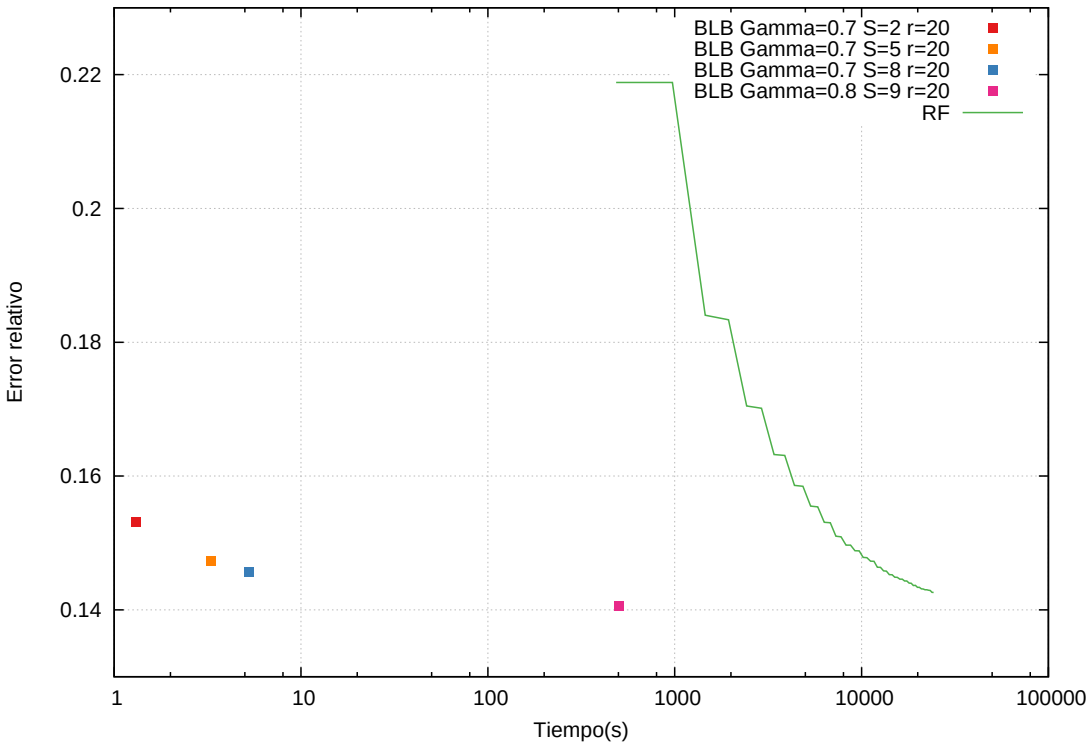


Figura 5.10: Error de test vs tiempo de training (s) para distintos valores de los hiperparámetros. Tamaño de las submuestras s para $\gamma = 0.7$ es de 14722 ejemplos (1.6 %); para $\gamma = 0.8$ es de 57995 ejemplos (6.4 %).

En la figura 5.10 se han representado el error frente al tiempo de entrenamiento de algunas ejecuciones de BLB y de Random Forest. Comprobamos que el BLB es mucho más rápido que Random Forest para este conjunto, y que además obtiene unos valores muy reducidos del error. De hecho, observamos que en un tiempo similar al que Random Forest emplea para calcular el primer árbol, el BLB es capaz de obtener un error menor que el correspondiente a la ejecución completa del Random Forest (50 árboles):

Dado los buenos resultados obtenidos para el waveform de 20000 ejemplos, y siendo este de la misma naturaleza pero mucho más grande, era de esperar obtener unos resultados iguales o mejores para el waveform con 1000000 ejemplos. Comprobamos que así es, obteniendo un error realmente bueno en tiempos radicalmente más bajos que con Random Forest.

El algoritmo BLB se ha desarrollado enfocado a ser aplicado sobre conjuntos de datos muy grandes, donde es muy probable que muchos ejemplos sean muy similiares entre sí y apenas estén aportando nueva información, como en este caso. Si el conjunto de datos es suficientemente grande puede suceder que con una fracción de los datos tengamos toda la información necesaria para aprender las reglas de clasificación del conjunto.

Error obtenido con BLB para $\gamma = 0.8; s = 9; r = 20$: **0.140547** en **503.06s**.

Error obtenido con Random Forest tras 50 árboles: **0.14259** en **24281.8s**.

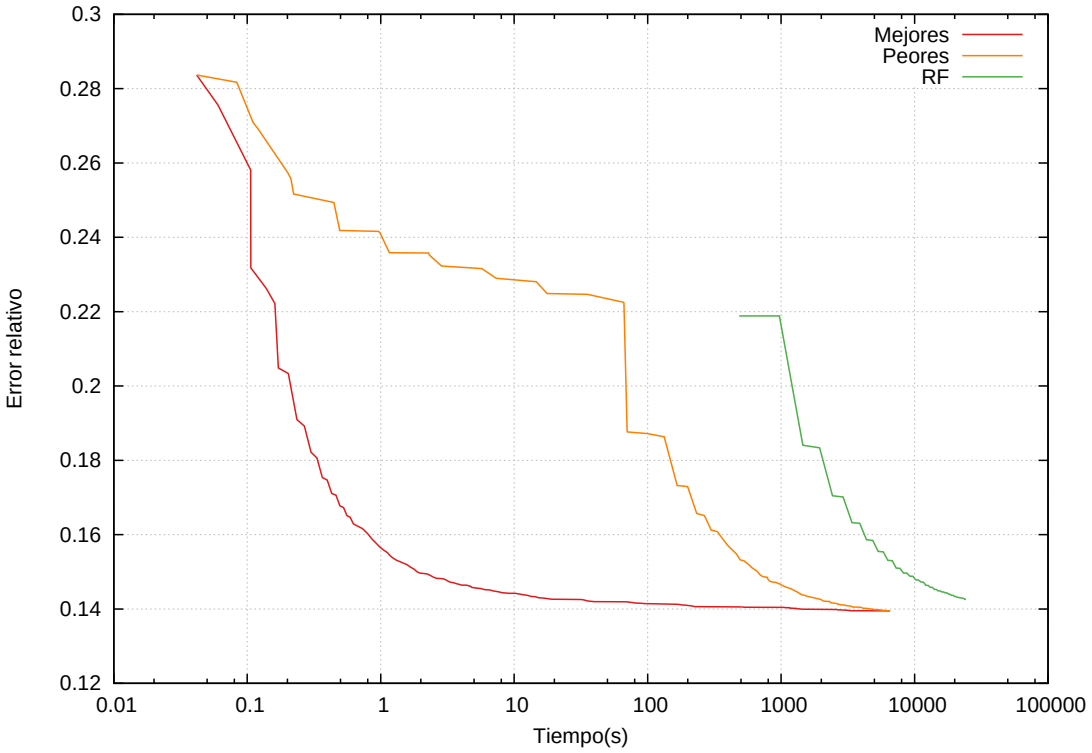


Figura 5.11: Comparativa mejores y peores ejecuciones vs Random Forest (Waveform 1000000 ejemplos)

En la figura 5.11 se representan las mejores y las peores ejecuciones del BLB junto con la ejecución de Random Forest. Observamos que ambas curvas de las ejecuciones de BLB se separan aún más de la curva de Random Forest, comparadas con lo que obtuvimos para waveform de 20000 ejemplos. Esto quiere decir que incluso aunque hagamos una elección de los hiperparámetros realmente mala, el BLB será capaz de mejorar el error de Random Forest en un tiempo inferior.

6

Conclusiones

En este trabajo hemos estudiado el algoritmo BLB: su funcionamiento, su eficiencia, sus parámetros, sus puntos fuertes, sus debilidades, etc.

Lo que propone éste algoritmo es coger pequeñas muestras de los datos y modificarlas ligeramente aplicando bagging para tratar de generar más muestras que acaben cubriendo el espectro de ejemplos que estamos descartando. Esto supone dos grandes ventajas: por un lado manejamos subconjuntos de datos mucho más pequeños que el conjunto de datos original, y por otro lado resulta mucho más económico en términos de tiempo generar un árbol nuevo a partir de los datos que ya hemos cargado que hacerlo cargando datos nuevos. Es decir, BLB es una aproximación de bagging más eficiente para ciertos conjuntos de datos.

Hemos observado que este algoritmo obtiene buenos resultados cuando el conjunto de datos es muy grande. Esto podría deberse a que para conjuntos suficientemente grandes una fracción de los datos puede contener toda la información necesaria para aprender las reglas de clasificación del conjunto.

Otra ventaja de BLB reside en que éste método es mucho más eficiente a la hora de ser paralelizado en el mundo Big Data ya que no es necesario enviar a cada worker del clúster el dataset completo sino sólo los pequeños subconjuntos s, a partir de los cuales va a realizar las r iteraciones. Además, al ser las submuestras mucho más pequeñas que el conjunto de datos original, los datos se pueden cargar en memoria con mayor facilidad; reduciendo los accesos a disco que ralentizan el proceso cuando el dataset original es más grande que la memoria de cada nodo.

Por último, cabe destacar que hemos comprobado que el método es muy robusto frente a la elección de los hiperparámetros s y r, pero los resultados varían más al hacerlo γ. Además, en la publicación original se recomienda el uso del $\gamma = 0.7$, pero

hemos comprobado que en los conjuntos estudiados variar éste parámetro puede mejorar notablemente los resultados.

En resumen, no parece que el BLB pueda reemplazar al bagging de forma genérica, sin embargo sí que puede ser una alternativa mucho más eficiente en muchos casos, principalmente para conjuntos muy grandes de datos.

7

Trabajo futuro

En este trabajo hemos tratado de cubrir algunos puntos clave del análisis del algoritmo BLB, pero existen muchos otros aspectos cuyo estudio podría ser interesante. Estos son algunos de ellos:

- Encontrar algún método para **estimar qué conjuntos son buenos candidatos para aplicar BLB**. Hemos visto que el BLB puede ser una alternativa eficiente a otros métodos de aprendizaje automático, sin embargo los resultados varían en función del conjunto de datos; por lo que sería interesante encontrar alguna herramienta que nos permita determinar si un conjunto es un buen candidato o no para ser modelizado con el BLB.

- **Paralelizar el BLB** para obtener mejores resultados. Como hemos comentado anteriormente, uno de los puntos fuertes del BLB es su facilidad para ser paralelizado; por lo tanto trasladar el algoritmo a un ecosistema de computación paralelizada, como por ejemplo Hadoop, permitiría una reducción del tiempo de ejecución drástica.

- **Optimizar tamaño de las muestras** r, por ejemplo empleando la estimación oob. Durante todo el proyecto las muestras r que hemos obtenido a partir de las submuestras s han sido generadas del mismo tamaño que el dataset original (N). Encontrar el valor óptimo (o al menos uno mejor) para el tamaño de resmuestro sería una posible vía para mejorar los resultados del BLB.

- Obtener los **remuestreos empleando una distrubión multinomial**. En vez de generar los remuestreos a partir de las submuestras extrayendo N elementos con repetición; es posible que resulte computacionalmente más eficiente generarlos empleando un algoritmo que genere la distribución equivalente sin necesidad de

realizar las N extracciones. Inicialmente habría que encontrar el algoritmo que al aplicarlo sobre todos los elementos del conjunto inicial generara el conjunto extendido; y posteriormente habría que valorar si resulta computacionalmente rentable. Es posible que para conjuntos de datos pequeños pudiera no resultar rentable pero para otros más grandes sí; debido a la diferencia de tamaño entre s y N.

- Estudiar **diferencias en los tiempos de test**. Hemos estudiado las diferencias en los tiempos de entrenamiento entre el BLB y Random Forest; sin embargo, otro aspecto a tener en cuenta sería la diferencia en los tiempos de test.

Bibliografía

[1] R. Kohavi y F. Provost. «Glossary of terms». En: Machine Learning 30 (1998), págs. 271-274.

[2] B. Efron y R. Tibshirani. «An Introduction to the Bootstrap». En: (1993).

[3] L. Breiman. «Bagging predictors». En: Machine Learning 26 (1996), págs. 123-140.

[4] L. Breiman. «Arcing classifiers». En: Ann. Statist. 26 (1998), 801–849.

[5] G. Martinez-Muñoz y A. Suárez. «Out-of-bag estimation of the optimal sample size in bagging». En: *Pattern Recognition* 43(1) (2010), págs. 143-152.

[6] Hall P. y Samworth R.J. «Properties of bagged nearest neighbour classifiers». En: *Journal of the Royal Statistical Society Series B* 67 (3) (2005), 363–379.

[7] Y. Freund y R.E. Schapire. «A decision-theoretic generalization of on-line learning and an application to boosting». En: *Proc. of 2nd European Conference on Computational Learning Theory* (1995), págs. 23-37.

[8] T. Dietterich. «An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization». En: *Machine Learning* 40 (2) (1999).

[9] L. Breiman. «Random Forests». En: *Machine Learning* 45(1) (2001), págs. 5-32.

[10] Sarkar P. Kleiner A. Talwalkar A. y Jordan M. I. «The Big Data Bootstrap». En: *ICML* (2012).

Apéndices

A

Conjuntos de datos estudiados

A.1. Breast Cancer Wisconsin (Original) Data Set

Resumen: Predecir la naturaleza de un tumor desarrollado a partir de células mamarias

Tarea: Clasificación

Ejemplos del conjunto: 699

Atributos: 9 más la clase

Missing values: Sí, 16 de los ejemplos contienen un valor desconocido

Descripción: Los ejemplos de este conjunto describen distintas propiedades observadas en células obtenidas de tumores desarrollados a partir de células mamarias. El objetivo final es predecir, a partir de los atributos observados, si el tumor es benigno o maligno. Estos datos se han ido generando periódicamente, procedentes de los casos clínicos del Dr. Wolberg; entre enero de 1989 y noviembre de 1991; sin embargo, trataremos todos los datos como un único conjunto sin tener en cuenta su fecha de obtención.

Información de los atributos:

1. Grueso del grano: 1 - 10
2. Uniformidad del tamaño de la célula: 1 - 10
3. Uniformidad de la forma de la célula: 1 - 10
4. Adhesión marginal: 1 - 10

5. Tamaño de una célula epitelial: 1 - 10
6. Núcleo aislado: 1 - 10
7. Cromatina blanda: 1 - 10
8. Nucleolo normal: 1 - 10
9. Mitosis: 1 - 10
10. Clase: (2 para benigno y 4 para maligno)

Comentar que en la muestra original se presenta un atributo más, el "Código identificador de la muestra". Sin embargo, para nuestro objetivo este atributo no aporta información relevante, por lo que decidimos suprimirlo.

Distribución de la clase:

2: Tumor benigno, 458 (65.5

4: Tumor maligno, 241 (34.5

A.2. MAGIC Gamma Telescope Data Set

Resumen: Clasificar si una señal detectada por un telescopio Cherenkov (el MAGIC) se corresponde con una radiación Gamma o no

Tarea: Clasificación

Ejemplos del conjunto: 19020

Atributos: 10 más la clase

Missing values: No

Descripción: Los datos a analizar se han generado empleando el método de Montecarlo para simular el registro de partículas gamma de alta energía en un telescopio situado Cherenkov, situado en la tierra, bajo la atmósfera. Este tipo de telescopios observan rayos gamma de alta energía, detectando la radiación que emiten las partículas cargadas producidas en las "duchas" electromagnéticas iniciadas por los rayos gamma. Esta radiación Cherenkov atraviesa la atmósfera y queda registrada en el detector, lo que permite reconstruir las características de la "ducha". La información que se recopila consiste en pulsos generados por los fotones Cherenkov que entran en unos tubos multiplicadores. En función de la energía de la radiación gamma inicial, pueden detectarse desde unos pocos cientos de fotones hasta unos 10000, obteniendo un patrón, denominado imagen de "ducha". Analizando los parámetros de esta imagen se puede discriminar estadísticamente los fotones que han sido producidos por radiaciones gamma, y cuales han sido producidos por "duchas" hadrónicas generadas por rayos cósmicos en

las partes más elevadas de la atmósfera de la Tierra. Una vez se ha obtenido la primera imagen, ésta se procesa aplicando PCA (Principal Component Analysis), generando una imagen con forma de elipse y cuyos parámetros vamos a analizar (comúnmente llamados parámetros de Hillas).

Información de los atributos:

Todos los parámetros que definiremos a continuación (exceptuando la clase) se corresponden con valores continuos y serán representados con números decimales de hasta 4 cifras decimales de precisión

1. fLength: longitud en mm del eje mayor de la elipse.
2. fWidth: longitud en mm del eje menor de la elipse.
3. fSize: el log decimal de la suma del contenido de todos los píxeles de la imagen.
4. fConc: el cociente entre la suma de los dos píxeles más altos y fSize.
5. fConc1: cociente entre el pixel más alto y fSize.
6. fAsym: distancia en mm del pixel más alto hasta el centro, proyectada en el eje mayor.
7. fM3Long: 3ra raíz del tercer momento a lo largo del eje principal (en mm).
8. fM3Trans: 3ra raíz del tercer momento a lo largo del eje menor (en mm).
9. fAlpha: ángulo del eje principal con el vector de origen (en grados).
10. fDist: distancia en mm desde el origen hasta el centro de la elipse.
11. class: g (señal gamma), h (hadrónica, es decir, ruido).

Distribución de la clase:

g = gamma (signal): 12332

h = hadron (background): 6688

En condiciones reales, la proporción de radiación gamma frente a la hadrónica sería mucho más baja, pero se ha modificado para mejorar la detección de la clase objetivo.

A.3. Poker Hand Data Set

Resumen: Predecir manos de póker

Tarea: Clasificación

Ejemplos del conjunto: 1000000

Atributos: 10 más la clase

Missing values: No

Descripción: Los ejemplos de este conjunto de datos describen las 5 cartas de una mano de póker. El objetivo es clasificar las manos en 10 posibles categorías:

0. Nada emparejado: carta alta.
1. Una pareja: dos cartas del mismo valor y distinto palo.
2. Doble Pareja: dos parejas de cartas del mismo valor.
3. Trío: tres cartas del mismo valor.
4. Escalera: cinco cartas con valores consecutivos seguidos. El As puede empezar o terminar las escaleras.
5. Color: cinco cartas del mismo color.
6. Full House: un trío y una pareja.
7. Póker: cuatro cartas del mismo valor.
8. Escalera de color: una escalera perteneciendo las cinco cartas al mismo palo.
9. Escalera real: una escalera de color empezando en el 10 y acabando en el As.

Cada carta queda descrita por su valor y su palo, por lo que cada ejemplo consiste en 10 atributos más la clase. Cabe destacar que el orden es importante, ya que por ejemplo, aunque sólo existen 4 posibles escaleras reales (una por cada palo), en nuestro conjunto de datos podremos encontrar cada una de ellas de 120 formas posibles (5!), haciendo un total de 480 posibles escaleras reales. Como características particulares de estos datos hay que resaltar que al ser una clasificación completamente determinista el error de Bayes es cero, por lo que a priori sería posible obtener una clasificación perfecta con error cero.

Información de los atributos:

- S1 "Palo de la primera carta". Atributo ordinal (1-4). Que representa los valores: Corazones, Picas, Diamantes, Tréboles
- C1 "Valor de la primera carta". Atributo numérico (1-13). Que representa los valores: (As, 2, ..., Reina, Rey)
- S2 "Palo de la segunda carta". Atributo ordinal (1-4). Que representa los valores: Corazones, Picas, Diamantes, Tréboles
- C2 "Valor de la segunda carta". Atributo numérico (1-13). Que representa los valores: (As, 2, ..., Reina, Rey)

- S3 "Palo de la tercera carta". Atributo ordinal (1-4). Que representa los valores: Corazones, Picas, Diamantes, Tréboles
- C3 "Valor de la tercera carta". Atributo numérico (1-13). Que representa los valores: (As, 2, ..., Reina, Rey)
- S4 "Palo de la cuarta carta". Atributo ordinal (1-4). Que representa los valores: Corazones, Picas, Diamantes, Tréboles
- C4 "Valor de la cuarta carta". Atributo numérico (1-13). Que representa los valores: (As, 2, ..., Reina, Rey)
- S5 "Palo de la quinta carta". Atributo ordinal (1-4). Que representa los valores: Corazones, Picas, Diamantes, Tréboles
- C5 "Valor de la quinta carta". Atributo numérico (1-13). Que representa los valores: (As, 2, ..., Reina, Rey)
- Clase "Mano de Póker". Variable ordinal (0-9)

Distribución de la clase:

El primero de los valores del paréntesis se corresponde con la distribución de la clase en los datos; el segundo valor es la probabilidad real teniendo en cuenta todas las posibles manos.

0. Nada emparejado: 12493 ejemplos, (49.95202 % / 50.117739 %).
1. Una pareja: 10599 ejemplos, (42.37905 % / 42.256903 %).
2. Doble Pareja: 1206 ejemplos, (4.82207 % / 4.753902 %).
3. Trío: 513 ejemplos, (2.05118 % / 2.112845 %).
4. Escalera: 93 ejemplos, (0.37185 % / 0.392465 %).
5. Color: 54 ejemplos, (0.21591 % / 0.19654 %).
6. Full House: 36 ejemplos, (0.14394 % / 0.144058 %).
7. Póker: 6 ejemplos, (0.02399 % / 0.02401 %).
8. Escalera de color: 5 ejemplos, (0.01999 % / 0.001385 %).
9. Escalera real: 5 ejemplos, (0.01999 % / 0.000154 %).

A.4. Waveform Database Generator (Version 1) Data Set

Resumen: Predecir el tipo de onda del ejemplo

Tarea: Clasificación

Ejemplos del conjunto: Al ser datos generados sintéticamente, podemos escoger cualquier valor arbitrario. Nosotros realizamos los experimentos con 20000 y con 1000000 datos.

Atributos: 21 más la clase

Missing values: No

Descripción: Se generan datos que se corresponden con 3 posibles ondas, y se les añade ruido (con media 0 y varianza 1) sobre los 21 atributos. Información de los atributos: Poner referencia al libro

Distribución de la clase: En el caso con 20000 datos la distribución es la siguiente:

- Clase 0: 6605 ejemplos (33.03 %)
- Clase 1: 6783 ejemplos (33.92 %)
- Clase 2: 6612 ejemplos (33.06 %)

En el caso con 1000000 datos la distribución es la siguiente:

- Clase 0: 332508 ejemplos (33.25 %)
- Clase 1: 333375 ejemplos (33.34 %)
- Clase 2: 334117 ejemplos (33.41 %)

Se puede apreciar que en ambos casos la distribución de las clases es muy homogénea.